CROSS-8 REFERENCE MANUAL

Universal Cross-Assemblers
P.O. Box 384
Bedford, Nova Scotia
Canada B4A 2X3

Copyright (C) 1985 by Universal Cross-Assemblers.
All rights reserved.

LICENSE

Use of this product by any person other than the purchaser or on more than one CPU at any specific time is strictly prohibited.

WARRANTY

Universal Cross-Assemblers will replace the Cross-8 source disk and/or reference manual once for a period of ninety days from the date of purchase if the original disk and/or reference manual are unreadable upon their return. The purchaser is urged to make and use a back-up copy of the source disk.

DISCLAIMER

Universal Cross-Assemblers, disclaims all liabilities to any damages or losses that may occur through the use of it's products or the breach of it's warranty, other than the single replacement of the original product.

ACKNOWLEDGEMENT

This program was made possible through the use of Turbo Pascal Version 3.0 which is the property of:

Borland Interational
4113 Scotts Valley Drive
Scotts Valley, CA, USA
95066

To Heather

Table of Contents

1 Introduction..... 5

2 Universal Truths Concerning Cross-8..... 6

3 The Cross-8 Command Line..... 7

4 The Cross-8 Assembly File 9

4.1 Assembly Line Format..... 9

4.2 Labels..... 10

4.3 Operands..... 11

4.3.1 Numeric Constants..... 11

4.3.2 String Constants..... 12

4.3.3 Arithmetic and Logical Operators... 12

4.4 Cross-8 Assembler Directives..... 14

4.4.1 CPU - Processor Declaration..... 14

4.4.2 DFB - Define byte..... 15

4.4.3 DWM - Define word most..... 16

4.4.4 DWL - Define word least..... 17

4.4.5 DFS - Define data storage..... 18

4.4.6 END - End of program..... 19

4.4.7 EQU - Equate..... 20

4.4.8 HOF - Hex Output Format..... 21

4.4.9 ORG - Origin..... 22

5 The Cross-8 Listing File 23

5.1 Listing Format..... 23

5.2 Assembly Error Codes..... 23

6 The Cross-8 Hexadecimal File 25

7 The Cross-8 Instruction Table 26

7.1 Operand Types..... 27

7.1.1 The A Operand Type..... 29

7.1.2 The B Operand Type..... 30

7.1.3 The C Operand Type..... 31

7.1.4 The E Operand Type..... 32

7.1.5 The I Operand Type..... 33

7.1.6 The L Operand Type..... 34

7.1.7 The P Operand Type..... 35

7.1.8 The R Operand Type..... 36

7.1.9 The S Operand Type..... 37

7.2 Register Definition..... 38

7.3 Sorting the Instruction Table..... 41

8 Non-Assembly Error Messages..... 43

8.1 Run Time Errors..... 43

8.2 I/O Errors..... 44

9 Processor Manufacturers..... 45

1 Introduction

Congratulations! You have selected one of the most cost effective tools for developing assembly language programs for many eight bit microprocessors and microcontrollers. But first a few words on assembly language programing in general.

An assembly language program is a text source file where manufacturer's mnemonics or "memory aids" are used to directly represent the binary codes that a computer's processor or brain actually executes. An assembler is a computer program that converts these mnemonics into a their corresponding binary codes. The simple word "assembler" usually refers to a resident or self-assembler, which assembles programs for the same processor it runs on. ASM, supplied with the CP/M-80 operating system is an example of this. A cross-assembler, assembles programs for a single processor which is different from it's host processor. Examples of these distributed by "the competition" may be found in the more technical computer and electronic journals.

Cross-8 (CRS8) from Universal Cross-Assemblers is actually a meta-assembler, in that it assembles programs for numerous different processors. It reads an assembly language source file (filename.ASM) and a corresponding sorted assembler instruction table (cpuname.TBL) from disk and writes an assembled listing (filename.PRN) and a hexadecimal machine language output file (filename.HEX). By using a flexible instruction table structure, CRS8 is designed to assemble machine language code for most microprocessors and microcontrollers with an 8 bit data word and an address word of 16 or less bits. To further enhance it's flexibility, CRS8 will produce a hexadecimal output file in the Intel, Motorola and Tektronix eight bit formats.

As one proceeds throught this manual, one will pass from the broad generalities needed to use CRS8, to the detailed specifics necessary to write an assembler instruction table. If the user simply wishes to assemble code for one of the processors with an instruction table provided, then the user need not read section 7 on creating an instruction table, but should carefully study the notes provided in the example given for that processor. To successfully tailor a provided table or create a new one, the user must study the entire manual, and perhaps several of the provided tables.

2 Universal Truths Concerning Cross-8

The following are broad generalities for those already familiar with compilers and assemblers. Others need not fret, all of the following will be explained in greater detail in the appropriate sections of this manual.

Cross-8 is a two pass cross-assembler with an optional third pass if a phase error is detected.

All input and output files will contain ASCII characters with each line terminated by an ASCII carrage-return and line-feed and each file terminated by a Control-Z.

The most significant bit of all ASCII characters is reset by CRS8 (i.e. ASCII_CHAR := ASCII_CHAR AND 01111111B).

Only one processor assembly instruction or CRS8 assembler directive is permitted per line.

All label declarations must be terminated with a colon ":", including labels for the EQU directive.

All labels, expressions and operands must have an integer value ranging from -32767 through 65535.

CRS8 makes no distinction between upper and lower case alphabetic characters.

Blank lines in the assembly source code (filename.ASM) will be reproduced in the assembly listing (filename.PRN) but otherwise ignored.

Overflow errors, undefined labels, and invalid expressions will be assigned a value of zero.

Labels must be less than 27 characters in length and only the first 9 of these are significant.

Cross-8 does not support conditional assembly or macros.

3 The Cross-8 Command Line

CRS8 may be run using the following command:

```
A>Q:CRS8 R:filename.SWU          Where...
```

A> Is the operating system prompt showing the default disk drive.

Q: Represents the label of the disk drive where the Cross-8 program (CRS8.COM) is located.

R: Represents the label of the disk drive where the assembly language source file may be found. This filename must appear in the disk directory as "filename.ASM".

S Represents the label of the disk drive where the assembled listing will be written. This filename will appear in the disk directory as "filename.PRN".

W Represents the label of the disk drive where the hexadecimal machine language file will be written. This filename will appear in the disk directory as "filename.HEX".

U Represents the label of the disk drive where the corresponding processor instruction table may be found. This file must appear in the disk directory as "cpuname.TBL". The "cpuname" is generated by the CPU directive in the assembly language source file (filename.ASM).

The output files from CRS8 may be directed to areas other than valid disk drives on your system using the following characters in the S and W locations:

X Will cause the .PRN or .HEX file to be written to the CON: device of your system, which is usually the video display.

T Will cause the .PRN or .HEX file to be written to the LST: device of your system, which is usually the printer.

Z Will cause the .PRN or .HEX file to be suppressed.

Therefore, the command

```
A>B:CRS8 A:T8085.XZB
```

will cause your system to execute the CRS8.COM file located on drive B to assemble the T8085.ASM file on drive A writing the T8085.PRN file to the CON: device and suppressing the T8085.HEX file using the cpuname.TBL file on drive B.

Any of the above input and output devices which are not specified will be assigned to the default disk drive. The minimum requirement to assemble a program is therefore:

```
A>CRS8 filename
```

In this case the cross-assembler will:

1) Expect to find CRS8.COM on the default disk drive.
2) Expect to find cpuname.TBL on the default disk drive.
3) Expect to find filename.ASM on the default drive.
4) Write fileame.PRN to the default drive.
5) Write filename.HEX to the default drive.

Running Cross-8 will produce a display similar to the following:

```
A>CRS8 T8085


CROSS-8    CP/M 80    Version - 1.20
Copyright (C) 1986 Universal Cross-Assemblers

Reading Assembly Source File...A:T8085.ASM
Reading Processor Table File...A:8085.TBL

Reading Assembly Source File...A:T8085.ASM
Writing Assembled List  File...A:T8085.PRN
Writing Assembled Hex   File...A:T8085.HEX

MEMORY AVAILABLE =  2.8174000000E+04 BYTES

A>
```

During execution, CRS8 informs the user of it's progress by displaying a message every time a file is opened. The assembly source file (.ASM), will of course be opened twice, once for each pass. The processor instruction table (.TBL) is only read once and kept in RAM for the second pass. Any assembly errors will be displayed on the CON: device (usually the video display) during the second pass, in addition to the listing (.PRN). Should the file specification not be included in the CRS8 command line, it will be prompted for using a single question mark "?" after the copyright message.

4 The Cross-8 Assembly File (.ASM)

The assembly file (.ASM) is the ASCII assembler source code to be assembled by CRS8. From the assembly file, the processor instruction file (.TBL) is selected, and the format of the hexadecimal file (.HEX) may be determined.

The following sub-sections describe the fundamental building blocks which make up the assembly source file. Examples are given for each section, but these have been assembled and taken from the listing file (.PRN); therefore the user does not provide the first 16 characters of these examples.

4.1 Assembly Line Format

Only one assembly instruction or assembler directive is permitted per line. The assembly line is free format, labels need not start in column 1 for example. Each line may contain some or all of the following sequence of identifiers:

```
line#     label:    operation     operand(s)     ; comment
```

Where...

Line# - is an optional decimal integer of range 0 to 32767 created by some editors representing the source code line number. If present, CRS8 will ignore this field except to reproduce it in the listing (.PRN).

Label - is a phrase starting with an alphabetic character or an underline "_" and ending with a colon ":", which is assigned the present value of the program counter or other user specified integer value. Characters within the label must be alphanumeric or an underline "_".

Operation - is a CRS8 assembler directive, or processor assembly instruction defined in the cpuname.TBL file specified by the CPU assembler directive. All operations must start with an alphabetic character.

Operand(s) - are labels, constants or expressions representing integer values in the range of -32767 through 65535 and/or character strings. They may be embodied within an assembly instruction.

Comment - is a statement following a semi-colon ";" usually used to describe the assembly program. The ";" may be placed anywhere on the assembly line and CRS8 ignores all characters following it on that line.

4.2 Labels

A label is an alphanumeric series of characters representing an integer in the range -32767 through 65535. The label field must start with an alphabetic ASCII character or a "_" and must end with a colon ":", even when used with the EQU assembler directive. Except for the first character, all remaining characters may be alphabetic, numeric or a "_". Except for the EQU directive, a label is optional and assigned the current value of the program counter. Labels may range in length from one to twenty-six characters, although only the first nine are significant. A label may stand alone on a line, in which case it will be assigned the current value of the program counter. Also, a dollar sign "$" may be used as an operand representing the current value of the program counter.

The following examples are taken from an actual Cross-8 listing. They use the equate (EQU) directive described further in section 4.4.7. Very simply, the equate (EQU) directive assigns the label on it's left, the value of the expression on it's right.

```
                ;Valid examples of labels are:
                ;
1234 =          STRT1:    EQU  1234H     ;Label on directive
1234 =          LD_UP:    EQU  1234H     ;Label with "_"
0000            alone:                   ;Stand alone label
0987 =          LONG_LONG_LONG: EQU 987H ;9 significant char.

0987 =          Shorter:  EQU  LONG_LONG_LONG
                                         ;But <= 26 will work

1234 =          _UN:      EQU  1234H     ;Starting with "_"


                ;Invalid examples of labels are:
                ;
S0000           1LABEL:   EQU  1234H     ;Starts with Number
S0000           MISS      EQU  1234H     ;Missing colon ":"
P1234 =         DUP:      EQU  1234H     ;Duplicate Label
P5678 =         DUP:      EQU  5678H     ;Duplicate Label
```

4.3 Operands

The following sub-sections describe numeric constants, string constants, and arithmetic and logical operators which in combination with labels may be used to form operands.

4.3.1 Numeric Constants

A numeric constant is a representation of a 16-bit integer in one of several number bases. All numeric constants must begin with a number, and their base is indicated by a trailing alphabetic character. The default base is base 10. The letter "Q" may also be used to specify octal numbers to avoid confusion between "O" and "0". The valid bases with their corresponding trailing character and character ranges are as follows:

B	Binary	Base 2	0-1
O	Octal	Base 8	0-7
Q	Octal	Base 8	0-7
D	Decimal	Base 10	0-9
H	Hexadecimal	Base 16	0-9, A-F

```
                ;Valid examples of numeric constants are:
                ;
00AF =          BIN1:     EQU  10101111B ;Binary
00FF =          OCT1:     EQU  377O      ;Octal
003C =          OCT2:     EQU  74Q       ;Octal
FFFF =          DEC1:     EQU  -1        ;Decimal
000A =          DEC2:     EQU  10D       ;Decimal
00FF =          HEX1:     EQU  0FFH      ;Hexadecimal

                ;Invalid examples of numeric constants are:
                ;
00000 =         HEX2:     EQU  0FFFFFH   ;Too large
L0000 =         BIN2:     EQU  00000000000000000000000000000000B
                                         ;Too long
L007B =         DEC3:     EQU  123.4     ;Illegal decimal
U0000 =         HEX3:     EQU  FFH       ;No leading number
L0025 =         OCT3:     EQU  37 7Q     ;Illegal Blank
```

4.3.2 String Constants

String constants consist of a series of ASCII characters between two single apostrophes ('). CRS8 will convert these constants to a hexadecimal representation of their ASCII values. Lower case characters within string constants will be represented as such. A single apostrophe (') cannot appear within a character string, for it will be interpreted as being the end of that string. If an apostrophe is needed, Universal Cross-Assemblers recommends that a double apostrophe be used ("). A string constant may also be used as an operand where applicable. In a DFB statement, a string constant must be 64 characters or less in length, not including the single apostrophes, but when used as an operand, or in the DWM, DWL, or EQU directives, an error will be flagged if the string constant exceeds two characters. A string constant cannot be extended beyond it's present line without terminating the constant with an apostrophe. Only the first five hexadecimal bytes representing the string constant will be shown in the listing (.PRN), however it will be fully represented in the hexadecimal file (.HEX).

In the following examples the define byte (DFB) directive is used as described in section 4.4.2. Very simply, the define byte (DFB) directive places the byte by byte value of the expression on the right into memory, starting at the present memory location shown on the left.

```
0000                    ORG  0000H
                ;Valid examples of string constants are:
                ;
0000 5965612C20         DFB  'Yea, yea, yea!'
000E 41426162           DFB  'AB','ab'
0012 5468652064         DFB  'The dog Said "Woof Woof!"'

                ;Invalid examples of string constants are:
                ;
E002B 00                DFB  "Hello"   ;Use single quotes
A002C 4D69737369         DFB  'Missing  ;Missing quote
```

4.3.3 Logical and Arithmetic Operators

CRS8 will accept a single logical or arithmetic operator as part of an operand expression. Any use of multiple operators or parentheses will result in an error. Since only one operator is permitted in any operand expression, there is no relative prescedence between operators. A list of recognized logical and arithmetic operators follows where x and y represent 16-bit integer values:

x OR y	Logical OR of x and y
x AND y	Logical AND of x and y
NOT y	Ones complement of y
x XOR y	Exclusive OR of x and y
x SHL y	Logical shift left of x by y bits and filled from the right with zeros
x SHR y	Logical shift right of x by y bits and filled from the left with zeros
< y	Binary value of low order byte only (y AND OFFH)
> y	Binary value of high order byte only (y SHR 8)
- y	Unary minus of y identical to 0-y or the twos complement of y
x + y	integer addition of x and y
x - y	integer subtraction of y from x
x * y	integer multiplication of x and y
x / y	integer division of x by y
x MOD y	integer remainder after division of x by y

```
                    ;Valid examples of operators are:
                    ;
00FF =              EXPRES1:  EQU  2 OR 253       ;Logical OR
0003 =              EXPRES2:  EQU  '3' AND 15     ;Logical AND
FFFA =              EXPRES3:  EQU  NOT 0101B      ;Ones complement
0011 =              EXPRES4:  EQU  10B XOR CR     ;Exclusive OR
3400 =              EXPRES5:  EQU  1234H SHL 8    ;Logical Shift Left
0012 =              EXPRES6:  EQU  1234H SHR 8    ;Logical Shift Right
0078 =              EXPRES7:  EQU  < 5678H        ;Low Byte
0056 =              EXPRES8:  EQU  > 5678H        ;High Byte
FFFF =              EXPRES9:  EQU  -1             ;Two's complement
4259 =              EXPRESa:  EQU  'AZ'+0FFH      ;Addition
0014 =              EXPRESb:  EQU  40-20D         ;Subtraction
4100 =              EXPREScc: EQU  'A' * 256      ;Multiplication
0042 =              EXPRESd:  EQU  'BA'/256       ;Division
0001 =              EXPRESe:  EQU  10 MOD 3       ;Remainder

                    ;Invalid examples of expressions are:
                    ;
L0009 =             EXPRESf:  EQU  3*3*3          ;2 operations
E0000 =             EXPRESg:  EQU  2**10D         ;Illegal operator
E0000 =             EXPRESh:  EQU  23/(-DIV)      ;Parantheses
```

4.4 Cross-8 Assembler Directives

CRS8 uses several common assembler directives combined with several of it's own as listed below.

CPU	CPU table declaration
DFB	DeFine Byte
DFS	DeFine data Storage
DWM	Define Word (Most significant byte first)
DWL	Define Word (Least significant byte first)
END	END of program
EQU	EQUate label to specified value
HOF	Hexadecimal Output Format declaration
ORG	ORiGin

4.4.1 CPU - CPU Table Declaration

The CPU directive instructs Cross-8 which processor assembly instruction table is to be accessed during assembly. The directive syntax is:

```
label:    CPU  cpuname   ;comment
```

Only the CPU filename may be specified with the CPU directive, the file type must be .TBL and the disk drive will be the default drive unless otherwise specified in the command line. Cross-8 will show the full name of the CPU file in the listing (filename.PRN). A label may be included with the CPU directive, which will be assigned the current value of the program counter, but this value will not be displayed in the listing. The CPU table is only read once by Cross-8 during assembly, and subsequent CPU directives will be ignored except for displaying the valid CPU filename. A nonexisting CPU filename will result in a fatal I/O error (see section 8.2). For further information on the CPU instruction file see section 7.

```
                ;Examples of the CPU directive
                ;
A:1802.TBL              CPU     1802    ; CPU TABLE
B:3870.TBL              CPU     3870    ; CPU TABLE
```

4.4.2 DFB - Define Byte

The DFB directive allows the user to define the value of storage areas on a byte by byte basis. The directive syntax is as follows:

```
label:    DFB  exprl,expr2,...,expr(n)  ;comment
```

Except for a string constant, the result of each expression must represent an 8 bit value or an error will be flagged. An expression may consist of a numeric constant, a string constant, a label, or an arithmetic or logical formula. The total hexadecimal value contained on a DFB line must not be greater than 80 bytes. Although only five bytes of this data will be shown in the listing, it will be included in the .HEX file in it's entirety.

```
 2000                       ORG  2000H
                 ;Valid examples of the DFB directive are:
                 ;
 2000 5061746965NOTE:       DFB  'Patient Lead Off',CR,LF
                                                ;ASCII string
 2012 000102                DFB  0,1,2          ;Integers
 2015 1548                  DFB  <$,77Q+9       ;Expressions
 2017 3F44                  DFB  3FH,44H        ;Hexadecimal

                 ;Invalid examples of the DFB directive are:
                 ;
B2019 FF                    DFB  03FFH          ;Too Large (10 Bits)
B201A 00                    DFB  $ * 999D       ;Overflow
```

4.4.3 DWM - Define Word (Most Significant Byte First)

The DWM directive allows the user to define the value of storage areas on a word by word basis (one word is two bytes). The total hexadecimal value contained on a DWM line must not be greater than 80 bytes. Although only five bytes of this data will be shown in the listing, it will be included in the .HEX file in it's entirety. The directive syntax is as follows:

```
label:    DWM  expr1,expr2,...,expr(n)  ;comment
```

The result of each expression must represent a 16 bit value or an error will be flagged. An expression may consist of a numeric constant, a string constant, a label, or an arithmetic or logical formula. Although ASCII string constants may be used, an error will be flagged if they are longer than two characters. The DWM directive will store the the most significant byte of the 16 bit value before the least significant byte. By using either the DWM or DWL directives, words may be placed in defined storage in a format which corresponds to the format used by the target processor.

```
 3000                       ORG  3000H
                 ;Valid examples of the DWM directive are:
                 ;
 3000 0000000100CONST1:     DWM  0,1,2      ;Numeric constants
 3006 3000                  DWM  CONST1     ;Label
 3008 00080048              DWM  <$,77Q+9   ;Expressions
 300C 003F0044              DWM  3FH,44H    ;Hexadecimal
 3010 FFFF                  DWM  0FFFFH     ;up to 16 bIts

                 ;Invalid examples of the DWM directive are:
W3012 0000                  DWM  'ABC'      ;Too Long
03014 0000                  DWM  $ * 999D   ;Too Large
```

4.4.4 DWL - Define Word (Least Significant Byte First)

The DWL directive allows the user to define the value of storage areas on a word by word basis (one word is two bytes). The total hexadecimal value contained on a DWL line must not be greater than 80 bytes. Although only five bytes of this data will be shown in the listing, it will be included in the .HEX file in it's entirety. The directive syntax is as follows:

```
label:    DWL  expr1,expr2,...,expr(n)  ;comment
```

The result of each expression must represent a 16 bit value or an error will be flagged. An expression may consist of a numeric constant, a string constant, a label, or an arithmetic or logical formula. Although ASCII string constants may be used, an error will be flagged if they are longer than two characters. The DWL directive will store the least significant byte of the 16 bit value before the most significant byte. By using either the DWM or DWL directives, words may be placed in defined storage in a format which corresponds to the format used by the target processor.

```
 4000                       ORG  4000H
                 ;Valid examples of DWL directive are:
                 ;
 4000 0000010002CONST2:     DWL  0,1,2      ;Numeric constants
 4006 06004800              DWL  <$,77Q+9   ;Expressions
 400A 3F004400              DWL  3FH,44H    ;Hexadecimal
 400E FFFF                  DWL  0FFFFH     ;up to 16 bits

                 ;Invalid examples of DWL directive are:
                 ;
W4010 0000                  DWL  'ABC'      ;Too Long
04012 0000                  DWL  $ * 7777Q  ;Too Large
```

4.4.5 DFS - Define Storage

The define storage (DFS) directive may be used to reserve a section of memory with unspecified contents during assembly. The directive has the following format:

```
label:    DFS  expression      ;comment
```

The expression can be of any form which represents a 16 bit value, but only one expression is allowed. The absolute value of the expression is added to the program counter and assembly continues. Except for the changed value of the program counter, no values are written to the hexadecimal file (.HEX).

```
5000                    ORG  5000H
                ;Valid examples of the DFS directive are:
                ;
5000            STOR_B:  DFS  1          ;Reserve for byte
5001            STOR_W:  DFS  3-1        ;Reserve for word

                ;Invalid examples of the DFS directive are:
                ;
W5003           STOR_E:  DFS  'ABC'      ;Not 16 bits
L5003           STOR_F:  DFS  3*5*7      ;Too many operators
```

4.4.6 END - End of Program

The end of assembly (END) directive is optional, but when used it will be the last line of the assembly file (.ASM) assembled (the remainder being ignored). This directive is of the following syntax:

```
label:    END  expression      ;comment
```

The expression is optional, but may represent any 16 bit value. When an expression is given, it's value will be included as the address in the final line of the hexdecimal machine language file, representing the starting address of the assembly program. If the END directive or expression are not included, the starting address will default to 0000H.

```
                ;Valid examples of the END directive are:
                ;
 9AC3                    END             ;Simple format
 C234           THE_END: END  RESET      ;With starting address
 C234                    END  0100H      ;Famous starting address

                ;An invalid example of the END directive is:
                ;
LC234                    END  1,2        ;Multiple operands
```

4.4.7 EQU - Equate

The equate (EQU) directive may be used to assign a 16 bit value to a label. The EQU format is as follows:

```
label:    EQU  expression     ;comment
```

Neither the label or the expression are optional in this directive. The expression may consist of any constant or formula whose value can be represented in 16 bits. Cross-8 will place the value of expression in the label field followed by an equal sign "=" to show that this value is not the current value of the program counter. Defining a label more than once, or placing multiple expressions after the EQU directive, will result in an error being flagged.

```
                ;Valid examples of the EQU directive are:
                ;
 0013 =         CR:       EQU  13H        ;ASCII Carrage Return
 000A =         LF:       EQU  10D        ;ASCII Line Feed
 5012 =         CNTR:     EQU  $          ;Program counter
 1300 =         EXPR2:    EQU  CR SHL 8   ;Formula

                ;Invalid examples of the EQU directive are:
                ;
L0021 =         TWICE:    EQU  33, -6     ;Multiple expressions
E0000 =         WHAT:     EQU             ;Missing expression
M0D05 =                   EQU  3333       ;Missing Label
```

4.4.8 HOF - Hexadecimal Output Format

The hexadecimal output format (HOF) selects the format of the hexadecimal machine language output file (filename.HEX). It has the following syntax:

```
label:    HOF  format    ;comment
```

Cross-8 will show the full name of the .HEX file in the listing (filename.PRN). A label may be included with the HOF directive, which will be assigned the current value of the program counter, but this value will not be displayed in the listing. The HOF directive is optional, and if not included Cross-8 will default to the INT format. The HOF directive may appear anywhere in the assembly source file (filename.ASM). If it appears more than once with different hexadecimal formats specified, the format of the hexadecimal file (filename.HEX) will change without an error code being generated.

```
                ;Valid examples of the HOF directive are:
                ;
 A:TEST.HEX             HOF  INT       ;Intel hexadecimal
 A:TEST.HEX             HOF  MOT       ;Motorola hexadecimal
 A:TEST.HEX     HEX:    HOF  TEK       ;Tektronix hexadecimal

                ;Invalid examples of the HOF directive are:
                ;
HA:TEST.HEX             HOF  RCA       ;Illegal hex format
HA:TEST.HEX             HOF  BIN       ;Illegal hex format
```

4.4.9 ORG - Origin

The origin (ORG) directive allows the user to specify the value of the program counter during assembly. This directive has the following syntax:

```
label:    ORG  expression     ;comment
```

The ORG directive may be used as often as is desired, but Cross-8 will not flag areas that may be defined more than once in a single program. The expression is not optional in this directive and may consist of any constant or formula whose value can be represented in 16 bits. Cross-8 will place the new value of the program counter in the label field. Placing no or multiple expressions after the ORG directive, will result in an error being flagged.

```
                ;Valid examples of the ORG directive are:
                ;
0000            RESET:    ORG  0          ;A common begining
0100                      ORG  0100H      ;A famous start

                ;Invalid examples of the ORG directive are:
                ;
L8000           TWICE2:   ORG  8000H,33   ;Multiple expressions
E0000           WHERE:    ORG             ;Missing expression
```

5 The Cross-8 Listing File

CRS8 produces a listing file (.PRN) during the second pass of the assembly source file. The listing may be sent to a number of devices as described in section 3.

5.1 Listing Format

The listing is the original assembly source file, with 16 additional ASCII characters inserted at the beginning of each line. An assembly error code may be placed in the first column, but under normal circumstances it will be blank. The next four characters represent the hexadecimal value of the program counter. Following the program counter, is another blank, followed by the a the hexadecimal value of the assembly instruction or assembler directive, if present. This value will not be displayed after the fifth byte, but the remainder will be placed in the .HEX file.

The EQU, CPU, and HOF assembler directives are exceptions to these rules, and further information on them may be found in section 4.4 of this manual.

5.2 Assembly Error Codes

When CRS8 detects an error during assembly, a single character error code is placed in the first column of the assembly listing on the line in which the error occurred. The error code and assembled line are also displayed on the CON: device (usually the video display), so that the user does not have to search the listing (or even generate it) to check for assembly errors. As CRS8 searches the instruction table (.TBL), it will display the first error code generated for the given assembly line, even though other errors may occur. Therefore, certain combinations of instructions and format errors can produce a misleading error code. If the code does not seem to be relavent, look for other possible errors. Errors are included thoughout the directive and instruction examples in this manual, and the alphabetic error codes represent the following:

A ASCII Error - Illegal format of an ASCII character string, usually a missing apostrophe (').

B Byte Error - Value declared in a DFB directive cannot be fully represented in one byte.

C Constant Error - Character in numeric constant is not valid for the specified base.

D Data Error - Operand defined in instruction table (.TBL) cannot be fully represented in length of bit field specified.

E Expression Error - Unable to evaluate expression as written.

H Hexadecimal Error - Illegal hexadecimal output format (.HEX) specified in HOF directive.

L Line Error - Unexpected characters found in assembly line, (missing ';') or single word found which is greater than 26 characters long.

M Missing Label - Label not found for EQU directive.

O Overflow - Numeric value specified cannot be represented in 16 bits.

P Phase Error - Value of label changes with successive assembler passes. When a phase error occurs in the second pass CRS8 will display the following message;

PHASE ERROR! ATTEMPTING THIRD PASS.

and attempt a third pass. This is to accomadate address shifts from instructions that may be identified as zero page addressing because of an undefined label in the first pass, changing to a extended addressing instruction in the second pass when the label is defined.

R Relative Error - Relative addressing branch instruction is out of range.

S Syntax Error - Instruction or directive cannot be identified.

T Type E Error - Given operand is not within 'E' register range specified by instruction table.

W Word Error - Defined value cannot be fully represented in a sixteen bit word (usually caused by a character string).

6 The Cross-8 Hexadecimal File

Cross-8 supports the Intel, Motorola and Tektronix 8 bit hexadecimal formats. The Intel format will be produced by default, and any one of the three may be selected using the HOF directive (see section 4.4.8). Examples of each are provided below for the assembly source file SHOW.ASM.

```
                ; File SHOW.ASM
                ;
A:SHOW.HEX              HOF  INT
0100                    ORG  0100H
0100 4865786164BEGIN:   DFB  'Hexadecimal!' ;Character String
010C                    END  BEGIN
```

INT -- Intel (8 bit format)

```
:0C0100004865786164656369 6D616C217D
:00010001FE
```

MOT -- Motorola (8 bit format)

```
S00D0000413A53484F572E48455823
S10F0100486578616465636 96D616C2179
S9030100FB
```

TEK -- Tektronix (8 bit format)

```
/01000C0D4865786164656369 6D616C2189
/01000001
```

Output of the .HEX file is controlled by the CRS8 command line as disscussed in section 3.

7 The Cross-8 Instruction Table

The CRS8 instruction table (cpuname.TBL) defines the instructions and registers of the individual microprocessors and microcontrollers. If one is simply going to use one of the tables provided, this section of the manual may be skipped, but the example assembly file for the processor to be used should be studied carefully.

There are of course numerous examples of cpu-files included with CRS8, which may be studied to gain further insight into the process of creating one. In addition to using the provided cpu-files, Cross-8 provides many other options. The simplest, is that the provided cpu-files may be modfied to generate a cross-assembler that is more specific to a particluar processor, rather than one that is general to a family of processors. This is usually just a matter of deleting or adding several instructions. Therefore, as manufacturers produce new processors based on older processor families, the user's development software will not become obsolete. Another feature is the ability to alter the manufacturers assembly mnemonics to a format which better suits the user's specific needs or preferences. A single set of common instruction mnemonics could be developed for example, and by only changing the opcodes in the instruction table, these same mnemonics could be assembled into machine language for a number of different processors. Only instructions that are extremely hardware specific would have to be re-examined. Using similar methods, Cross-8 can be used to ease software upgrading to a more recent processor, especially if both processors are from the same manufacturer, and have similar instruction sets. Undocumented instructions, that have been found to exist in several common microprocessors may also be added.

The instruction table (cpuname.TBL) consists of an alphabetically ordered listing of all the processor's instructions, the format and placement of it's operands, and the corresponding hexadecimal machine language or opcode. The instruction table must be sorted by the supplied program CRS8SORT.COM before use. Each instruction must be on a separate line and followed by an ASCII carrage return and line feed. An operand may be placed anywhere within the instruction, with its characteristics enclosed in a set of script brackets "{}". The ASCII hexadecimal opcode for the instruction is placed between the carat "^" and colon ":". From this field CRS8 determines both the value and length of the instruction's opcode. An opcode cannot be longer than 8 bytes, or 16 ASCII hexadecimal characters. During assembly, the supplied opcode is converted to a binary value and logically ORed with any operands defined within the script brackets "{}". Following the last instruction an optional asterisk "*" may be used to signify the start of a section of equate (EQU) assembler directives, which are normally used to define register values. During assembly these are treated in an identical fashion to the an EQU directive in the assembly source file (see section 4.4.7). The syntax of the instruction file is as follows:

```
Instruction (operand(s)) ^ Hexstring :
      "          "        "       "     "
      "          "        "       "     "
      "          "        "       "     "
      "          "        "       "     "
      "          "        "       "     "

*
label:    EQU  expression
  "        "        "
  "        "        "
  "        "        "
```

7.1 Instructions Types

The instructions and operands are combined into a single expression on the left side of the "^". There must not be blanks anywhere in this field. Any blanks that may be specified in the processor's assembly language programming manual must be excluded. This will not prevent the user from placing blanks or tabs in the assembly language source file (filename.ASM).

Operands within an instruction are optional, and there is no set limit to the number there may be, except that the total length of the instruction field is limited to 33 ASCII characters.

For example, in the 8085.TBL instruction table, the enable interrupt instruction, which does not have any operands, appears as follows:

```
EI^FB:
```

When assembled by CRSB, a listing using this instruction appears as follows:

```
 6000                       ORG  6000H
                 ;Valid instructions without operands are:
                 ;
 6000 FB                    EI            ;Enable Interrupts
 6001 FB         Interr:    e i           ;Spaces do not matter

                 ;Invalid instructions without operands are:
                 ;
L6002                       EI MODEM      ;Illegal Operand
S6002                       IE            ;Syntax Error
```

As stated previously, an operand is defined by placing it in script brackets in it's proper location within the instruction. The operand definition, within the brackets has the following syntax:

{ Starting-Location Operand-Type Operand-Length }

Where:

Starting-Location is the position of the first bit of the operand within the instructions opcode. This value must be specified as a decimal number without a base indicator (D). The bit positions are numbered from left to right (Most significant bit first) starting at 0 and preceding to a maximum of 63.

Operand-Length is number of signifcant bits of the operand which will be placed in the opcode, beginning with the least significant bit. The operand length must be a deicmal value without a base indicator ranging from 1 to a maximum of 16.

Operand-Type is a single character which represents any changes that may be made to the opcode or checks that may be performed before it is placed there. The operand-types used in Cross-8 are as follows:

7.1.1 The A Operand Type

A The operand is ORed with the opcode as specified by the starting-location and operand-length. A range check is performed on the value of the operand during assembly, to insure that it's absolute value can be fully represented via a twos complement format in the number of bits specified by the operand-length. If it cannot, a data error 'D' is generated and Cross-8 will proceed further in the instruction table looking for an instruction which will handle the operand. If a matching instruction with a sufficiently long bit-length is found by Cross-8, the data error 'D' is not displayed and this new instruction is used. As an example, examine three "Rotate Memory Byte Left Through Carry" instructioons for the 6809 processor. (Note: A table for the 6809 processor is only included with the PC/MS-DOS version of Cross-8).

```
ROL(011A5),X^6900:
ROL(016A8),X^698800:
ROL(16A16),X^69890000:
```

In this instruction, an offset and the contents of the X-Register are added to determine the address of the byte to be rotated. Note that the bit-length of the offset (011A5) and the length of the opcode increases with each subsequent instruction. The first instruction will support an offset range of +- 15, the second +- 127, while the third supports an offset range of +- 32767. When an (ROL OFFSET,X) instruction is found in the source assembly file, Cross-8 will evaluate the value of the offset. If it can be represented via twos complement notation in 5 bits, the first instruction is assembled, else, if it can be represented in 8 bits the the second instruction is assembled, else, if it can be represented in 16 bits the third instruction is assembled, else, an error occurs. It is essential that a leading zero be placed in the starting-location of the instructions with the shorter opcodes to insure that they appear first after being sorted by CRSBSORT. This procedure will enable Cross-8 to automatically select the shortest opcode for the ROL OFFSET,X instruction.

```
7000                    ORG  7000H
                ;Instructions with absolute range checks are:
                ;
000C =          OFFSET1:  EQU 000CH     ;Small offset
0068 =          OFFSET2:  EQU 0068H     ;Medium offset
5529 =          OFFSET3:  EQU 5529H     ;Large offset

7000 690C               ROL OFFSET1,X ;Short opcode
7002 698868             ROL OFFSET2,X ;Longer opcode
7005 69895529           ROL OFFSET3,X ;Longest opcode
```

7.1.2 The B Operand Type

B The operand is ORed with the opcode as specified by the starting-location and operand-length. A range check is performed on the value of the operand during assembly, to insure that it can be fully represented in the number of bits specified by the operand-length. If it cannot, a data error 'D' is generated and Cross-8 will proceed further in the instruction table looking for an instruction which will handle the operand. If a matching instruction with a sufficiently long bit-length is found by Cross-8, the data error 'D' is not displayed and this new instruction is used. As an example, examine two AND memory with accumulator instructions for the 6801 processor.

```
ANDA(0BB8)^9400:
ANDA(BB8)^B40000:
```

The first instruction is of the direct addressing type, used with memory locations 0-0FFH. The second instruction is of the extended addressing type and may be used with memory locations 0-0FFFFH. When an (AND A) instruction is found in the source assembly file (filename.ASM) by Cross-8, it will check the length of the instruction's operand. If the operand is 8 or less bits in length, the first instruction will be assembled, but if the operand is 9 to 16 bits in length then the second instruction is assembled. It is essential that a leading zero be placed in the starting-location of the direct addressed instruction to insure that it appears first after being sorted by CR88ORT. This procedure will enable Cross-8 to automatically select the shortest opcode for the AND A instruction. It also enables many of the supplied tables to distinguish between register values and data locations.

```
 8000                    ORG  8000H
                 ;Valid instructions with bit range checks are:
                 ;
 0067 =          Page0:  EQU  0067H    ;Page zero
 1234 =          Not0:   EQU  1234H    ;Elsewhere
                 ;
 8000 9467               AND A  PAGE0  ;AND accumulator
 8002 B41234             AND A  NOT0   ;AND accumulator

                 ;Invalid instructions with bit range checks are:
                 ;
08005 9400               ANDA   1FFFFH ;Too large
W8007 9400               ANDA   'ABC'  ;Too long
```

7.1.3 The C Operand Type

C The operand is ORed with the opcode as specified by the starting-location and operand-length. This operand-type is identical to the B type except that a range check is not performed. It is necessary to assemble instructions that only make use of less significant bits of the operand. An example of this is the short branch (BR) instruction used by the 1802 series of microprocessors.

```
BR(8C8)^3000:
```

The short branch instruction loads the second byte of the instruction into the program counter, effectively creating a branch within the current 256 byte page. The value of the operand specified for the branch instruction in the assemtly source file may indeed be longer than 8 bits, which would have been incorrectly flagged as an error if the 'B' operand-type was specified.

```
 9000                    ORG  9000H
                ;A valid instruction without bit range checks is:
                ;
 9000 3057               BR   9057H    ;No Error Generated

                ;An invalid instruction without bit range  checks
is:
                ;
09002 3000              BR   1FFFFH    ;Too Long
```

7.1.4 The E Operand Type

E This operand type is used by the Z8 family to distinquish between RAM and register addresses. A range test is performed in which the value of the operand specified in the assembly language source file must be in the range on 0E0H to 0EFH. Should this not be the case a data error 'D' is generated and Cross-8 will continue through the cpufile looking for a matching instruction with a suitable operand-type. If the value of the specified operand is within the 'E' range, then the operand will be ORed with the opcode as specified by the starting-location and the operand-length. The use of the 'E' operand type may be seen in the exclusive-OR instructions used by the Z8 family. Notice the use of leading zeros to ensure that the register addressing instructions are examined before the direct addressing instructions.

```
XOR@(08B8),#(16B8)^B70000:
XOR(08B8),#(16B8)^B60000:
XOR(08E4),@(12E4)^B300:
XOR(08E4),(12E4)^B200:
XOR(16B8),@(8B8)^B50000:
XOR(16B8),(8B8)^B40000:
```

```
A000                    ORG 0A000H
                ;Instructions with E register addressing are:
                ;
0055 =          TEMP1:  EQU  0055H      ;Internal RAM
0056 =          TEMP2:  EQU  0056H      ;Internal RAM

A000 B21E               XOR  r1,R14     ;Register, register
A002 B31E               XOR  r1,@R14    ;Register, indirect
A004 B45655             XOR  TEMP1,TEMP2 ;Direct, direct
A007 B5E355             XOR  TEMP1,@R3  ;Direct, indirect
A00A B65635             XOR  TEMP2,#35H ;Direct, immediate
A00D B7E409             XOR  @R4,#1001B ;Indirect, immediate
```

7.1.5 The I Operand Type

I The high and low bytes are exchanged in a similar fashion to the DWL directive, and the result is ORed with the opcode as specified by the starting-location and operand-length. Since this operand type is used primarily with instructions that utilize the full sixteen bits of the operand, a range check is not performed.

```
JP(8I16)^C30000:
```

The Z80 is one processor that uses sixteen bit addresses with the least significant byte placed first in the opcode, as can been seen in the jump to address following (JP) instruction.

```
 B000                    ORG 0B000H
                ;Valid instructions with exchanged bytes are:
                ;
 B000 C32211             JP    1122H      ;Notice hex code
 B003 C34433             JP    3344H      ;Notice hex code

                ;Invalid instructions with exchanged bytes are:
                ;
0B006 C30000             JP   1FFFFH      ;Too Large
WB009 C30000             JP    'ABC'      ;Too Long
```

7.1.6 The L Operand Type

L A relative operand is calculated by Cross-8 during assembly and it's high and low bytes are exchanged in the same manner as the DWL directive and the I operand type. The result is ORed with the opcode as specified by the starting-location and operand-length. This relative value is the value of the operand specified in the assembly language source file, minus the memory address of the byte immediately following the opcode. A range check is performed during assembly, to ensure that the absolute value of the calculated operand is less than 2**(Operand-length). If this condition is not true, a range error 'R' will be flagged, and Cross-8 will look for a matching instruction with a longer operand-length.

LCALL(8L16)^CD0000:

This type of inverted relative addressing is used by the 8096 family of microcontrollers for their long call subroutine instruction.

```
 C000                    ORG 0C000H
                 ;Valid instructions with relative exchanged bytes are:
                 ;
 C000 EF85D9             LCALL 9989H      ;Notice hex code
 C003 EF8AB8             LCALL 7890H      ;Notice hex code

                 ;Invalid instructions with relative exchanged bytes are:
                 ;
0C006 EF0000             LCALL 1FFFFH     ;Too Large
WC009 EF0000             LCALL 'ABC'      ;Too Long
```

7.1.6 The P Operand Type

P The page operand-type is used primarily by the 8048 and 8051 family of microcontrollers for their eleven bit call and jump instructions. The page operand is similar to the 'C' operand type except that bits 10 through 8 of the operand specified in the assembly language source file are ORed with bits 0 through 2 of the opcode, respectively. The operand is also ORed with the opcode as specified by the starting-location and operand-length. Like the 'C' operand-type a range check is not performed.

ACALL(8P8)^1100:

An example of the page operand-type is the absolute call instruction of the 8051 family of microcontrollers.

```
 D000                    ORG 0D000H
                ;An instruction with page addressing is:
                ;
 D000 3123               ACALL 0D123H ;Absoulute call
```

7.1.8 The R Operand Type

R A relative operand is calculated by Cross-8 during assembly, and the result ORed with the opcode as specified by the starting-location and operand-length. This relative value is the value of the operand specified in the assembly language source file, minus the memory address of the byte immediately following the opcode. A range check is performed during assembly, to ensure that the absolute value of the calculated operand is less than 2**(Operand-length). If this condition is not true, a range error 'R' will be flagged, and Cross-8 will look for a matching instruction with a longer operand-length. This procedure will enable Cross-8 to automatically select the shortest opcode for an instruction. Many processors use different mnemonics for the relative and absolute branching instruction. The user may wish to make these different mnemonics similar so that Cross-8 will select the shortest mnemonic.

BCC(8R8)^2400:

An example of an instruction that uses relative addressing, is the branch on carry clear instruction used by the 6805 micrcontroller.

```
 E000                    ORG 0E000H
                 ;A valid instruction with relative addressing is:
                 ;
 E000 2400               BCC  SUB1      ;Branch on carry clear
 E002            SUB1:   DFS  260D      ;Subroutine 1
 E106            SUB2:                  ;Subroutine 2

                 ;An invalid instruction is:
                 ;
RE106                    BCC  SUB1      ;Too far
```

7.1.9 The S Operand Type

S This operand-type is identical to the 'R' type except than the relative operand is calculated as the operand value specified in the assembly source file minus the address of the last byte of the instruction. The relative operand is calculated by Cross-8 during assembly, and the result ORed with the opcode as specified by the starting-location and operand-length. A range check is performed during assembly, to ensure that the absolute value of the calculated operand is less than 2**(Operand-length). If this condition is not true, a range error 'R' will be flagged, and Cross-8 will look for a matching instruction with a longer operand-length. This procedure will enable Cross-8 to automatically select the shortest opcode for an instruction. Many processors use different mnemonics for the relative and absolute branching instruction. The user may wish to make these different mnemonics similar so that Cross-8 will select the shortest mnemonic.

BNO(8S8)^9800:

An example of an instruction that uses this operand-type is the "Branch If No Overflow" instruction used by the 3870 family.

```
 F000                    ORG 0F000H
                ;A valid example of alternative branching is:
                ;
 F000 9821              BNO  0F022H   ;Branch if no overflow

                ;An invalid example of alternative branching is:
                ;
0F002 9801              BNO  01000H     ;Too far
```

7.2 Register Definition

An asterisk "*" may be placed on the line following the last assembly instruction in the instruction table. On subsequent lines, standard equate (EQU) directives may be used to define register values or whatever other labels the user may wish. For further information on the EQU directive see section 4.4.7. This definition section is optional, and may be placed in the assembly language source file instead.

As an example, the 8085 instruction table file (8085.TBL) showing the asterisk "*" and register definitions after the instruction table, is shown on the next two pages.

For further examples of instruction tables examine the .TBL files on the CRS8 source disk.

```
ACI(8B8)^CE00:
ADC(5B3)^88:
ADD(5B3)^80:
ADI(8B8)^C600:
ANA(5B3)^A0:
ANI(8B8)^E600:
CALL(8I16)^CD0000:
CC(8I16)^DC0000:
CMA^2F:
CMC^3F:
CMP(5B3)^B8:
CM(8I16)^FC0000:
CNC(8I16)^D40000:
CNZ(8I16)^C40000:
CPE(8I16)^EC0000:
CPI(8B8)^FE00:
CPO(8I16)^E40000:
CP(8I16)^F40000:
CZ(8I16)^CC0000:
DAA^27:
DADB^09:
DADD^19:
DADH^29:
DADSP^39:
DCR(2B3)^05:
DCXB^0B:
DCXD^1B:
DCXH^2B:
DCXSP^3B:
DI^F3:
EI^FB:
HLT^76:
INR(2B3)^04:
INXB^03:
INXD^13:
INXH^23:
INXSP^33:
IN(8B8)^DB00:
JC(8I16)^DA0000:
JMP(8I16)^C30000:
JM(8I16)^FA0000:
JNC(8I16)^D20000:
JNZ(8I16)^C20000:
JPE(8I16)^EA0000:
JPO(8I16)^E20000:
JP(8I16)^F20000:
JZ(8I16)^CA0000:
LDAXB^0A:
LDAXD^1A:
LDA(8I16)^3A0000:
LHLD(8I16)^2A0000:
LXIB,(8I16)^010000:
LXID,(8I16)^110000:
LXIH,(8I16)^210000:
LXISP,(8I16)^310000:
```

```
MOV(2B3),(5B3)^40:
MVI(2B3),(8B8)^0600:
NOP^00:
ORA(5B3)^B0:
ORI(8B8)^F600:
OUT(8B8)^D300:
PCHL^E9:
POPB^C1:
POPD^D1:
POPH^E1:
POPPSW^F1:
PUSHB^C5:
PUSHD^D5:
PUSHH^E5:
PUSHPSW^F5:
RAL^17:
RAR^1F:
RC^D8:
RET^C9:
RIM^20:
RLC^07:
RM^F8:
RNC^D0:
RNZ^C0:
RP^F0:
RPE^E8:
RPO^E0:
RRC^0F:
RST(2B3)^C7:
RZ^C8:
SBB(5B3)^98:
SBI(8B8)^DE00:
SHLD(8I16)^220000:
SIM^30:
SPHL^F9:
STAXB^02:
STAXD^12:
STA(8I16)^320000:
STC^37:
SUB(5B3)^90:
SUI(8B8)^D600:
XCHG^EB:
XRA(5B3)^A8:
XRI(8B8)^EE00:
XTHL^E3:
*
B: EQU 0
C: EQU 1
D: EQU 2
E: EQU 3
H: EQU 4
L: EQU 5
M: EQU 6
A: EQU 7
```

7.3 Sorting the Instruction Table

After being entered or modified, the instruction table (.TBL) must be sorted using CRS8SORT.COM. This program will search the instruction table for errors, and finding none will sort the instructions alphabetically and display the resulting file on the CON: device (usually the video screen). The error instructions are detailed and include the instruction at fault, and therefore self explanatory.

CRS8SORT may be run using the following command:

A>Q:CRS8SORT R:cpuname Where...

A> Is the operating system prompt showing the default disk drive.

Q: Represents the label of the disk drive where the Cross-8 sort (CRS8SORT.COM) program is located.

R: Represents the label of the disk drive where the instruction table to be sorted may be found. This filename must appear in the disk directory as "cpuname.TBL". After a successful sort, the sorted instruction table will be written under it's original name to the device from which it came.

The sorted instruction file from CRS8SORT may not be directed to areas other than the device from which it was read.

Therefore, the command

A>B:CRS8SORT A:COP400

will cause your system to execute the CRS8SORT.COM file located on drive B to load and sort the COP400.TBL file on drive A, writing a successfully sorted COP400.TBL instruction table back to drive A.

Any of the above devices which are not specified will be assigned to the default disk drive. If a filename is not included when CRS8SORT is run, the user will receive a prompt for it via a question mark "?" following the copyright message. The minimum requirement to sort an instruction table is therefore:

```
A>CRS8SORT cpuname
```

From this command your computer will:

1) Expect to find CRS8SORT.COM on the default disk drive.
2) Expect to find cpuname.TBL on the default disk drive.
3) Rewrite a successfully sorted instruction table under cpuname.TBL on the default disk drive.

8 Non-Assembly Error Messages

During the assembly of a program certain errors will cause CRS8 to terminate execution. When this occurs, all open files are closed, an error message is sent to the CON: device (usually a video monitor) and CRS8 exits to the operating system. The following example illustrates an I/O error when CRS8 tries to read an assembly source file which does not exist.

```
A>CRS8 NOFILE


CROSS-8    CP/M 80    Version - 1.20
Copyright (C) 1986 Universal Cross-Assemblers

Reading Assembly Source File...A:NOFILE.ASM

?NOFILE.

I/O error 01, PC = 2380
Program aborted

A>
```

8.1 Run Time Error Messages

A number of events during assembly will cause CRS8 to abort and produce a run-time error. When this occurs, the last assembly line read by CRS8 will be displayed on the CON: device (usually the video screen) following a question mark "?". Below this line an error code and error location will be displayed, similar to the example in section 8. By examining the line and error code, mistakes should be easily found. The error code will be one of the following types:

01	Floating point overflow.
02	Division by zero attempted.
04	Bit length in .TBL negative or zero.
10	String length error.
92	Out of integer range.
FF	Processor table too large and/or too many labels.

8.2 I/O Error Messages

A number of I/O events during assembly will cause CRS8 to abort and produce an I/O error. The last assembly line read by CRS8 will be displayed on the CON: device (usually the video screen) following a question mark "?". Below this line an error code and error location will be displayed, similar to the example shown in section 8. By examining the ouptut line, the error code, and the file status updates produced by CRS8, errors should be easily found. The displayed error code will be one of the following types:

01	File does not exist.
99	Unexpected end-of-file (Missing Ctrl-Z).
F0	Disk is full.
F1	Disk directory is full.
F2	File too large.
FF	File disappeared.

9 Processor Manufacturers

The following is an alphabetical list of the manufacturers from whom further information may be obtained regarding the microprocessors and microcontrollers for which instruction tables are supplied by Universal Cross-Assemblers:

Intel Corp 3065 Bowers Avenue Santa Clara, CA 95051	8048 8051 8085 8096
Mostek Corp 1215 W Crosby Road Carrollton, TX 75006	3870
Motorola Integrated Circuits 3501 Ed Bluestein Blvd Austin, TX 78721	6801 6805 6809
National Semiconductor Corp 2900 Semiconductor Drive Santa Clara, CA 95051	COP400
RCA Solid State Division Rte 202 Somerville, NJ 08876	1802
Rockwell International Microelectronic Devices Div 4311 Jamboree Rd Newport Beach, CA 92660	6502
Zilog Inc 1315 Dell Ave Cambell, CA 95008	Z8 Z80

NOTES

C R O S S - 8

MAYBE THE ONLY DEVELOPMENT SOFTWARE YOU'LL EVER NEED

Cross-8 from Universal Cross-Assemblers is a flexible, table based cross-assembler designed for anyone who does not wish to purchase an expensive, dedicated microprocessor development system. It is ideal for the hobbyist or firm that changes processors frequently for different designs. It can also reduce capital costs by saving your expensive emulator/development system strictly for debugging, while programmers assemble code on their PC's using Cross-8. Since Cross-8 will produce hexadecimal machine language files in the three most popular 8 bit formats, it can be used with almost any EPROM programmer or emulator.

Cross-8 reads an assembly language source file and a corresponding sorted assembler instruction table from disk and writes an assembled listing and a hexadecimal machine language output file. By using a flexible instruction table structure, Cross-8 is designed to assemble machine language code for most microprocessors and microcontrollers with an 8 bit data word and an address word of 16 or less bits.

Assembly Language Input

The assembly language source file may be prepared using any ASCII text editor, but must be listed on the disk directory as filename.ASM. Cross-8 will ignore the most significant bit of all ASCII characters, which are set by some word processors. Cross-8 will select the corresponding processor instruction table (cpuname.TBL) using the CPU directive within the source code. An assembled listing will be produced, appearing in the disk directory as filename.PRN. The hexadecimal machine language file will appear listed as filename.HEX, and its format may be specified as either the Intel, Motorola, or Tektronix 8 bit formats using the HOF directive in the assembly source file. The disk drive from which all files will be read or written may be specified in the Cross-8 command line, or will default to the active drive. The listing and hexadecimal files may optionally be sent to the screen or printer or be suppressed. Assembly errors are marked with one of 14 different alphabetic codes in the first column

of the listing, and will be displayed on the screen so that the listing does not have to be searched.

Cross-8 supports the following assembler directives:

CPU — Processor declaration
DFB — Define byte
DFS — Define data storage
DWM — Define word (High byte : Low byte)
DWL — Define word (Low byte : High byte)
END — End of assembly
EQU — Equate
HOF — Hex output format declaration
ORG — Origin
; — Comment

Within the source code, all label declarations must be terminated with a colon ":", but the assembly line is free format, so labels do not have to start in column 1. Labels must be less than 27 characters in length and only the first 9 of these are significant. An optional integer line number may appear at the beginning of each assembly line. Numeric constants may be specified with their bases declared using one of the trailing alphabetic characters B, O, Q, D or H with the default base being decimal. String constants may also be defined between two apostrophes ('). A single operator may be used in the declaration of an operand from one of the following: OR, AND, NOT, XOR, SHL, SHR, MOD, <, >, +, -, /, *. A dollar sign $ represents the current value of the program counter. Blank lines will be ignored but repeated in the assembled listing.

Processor Instruction Tables

Full instructions for writing new tables or modifying existing ones for the processor of your choice are included in a 36 page manual. The program CRS8SORT.COM is also included to sort your instruction tables and check them for format errors before use. Commonly used labels, such as those used for registers and I/O ports, may also be defined in the instruction table. Instruction tables and example assembly source files with notes on particular processors are included for the following families of processors:

1802	6805	8096
3870	8048	COP400
6502	8051	Z8
6801	8085	Z80

Cross-8 also gives users the ability to create their own cross-assembler for almost any 8 bit processor. Therefore as new processors are introduced, the user simply has to write a new table to assemble programs for them. Since many of the new processors have instruction sets based on existing processor families, this can be as simple as adding several lines to an existing table.

The true hackers may even wish to write their own assembly instruction set for an existing processor, combine several opcodes to form new instructions, or modify an existing instruction set to their personal preferences. A single set of common instruction mnemonics could even be developed for many processors, and only the hardware specific instructions would have to be re-examined when assembling programs for different environments.

Technical Information

Cross-8 is about 25K long, requires 17 bytes per label, and 52 bytes for each instruction in the table. For CP/M-80 systems, a minimum of 64K RAM is recommended, for PC/MS-DOS systems at least 128K RAM is recommended. Two disk drives are not necessary.

Cross-8 is not copy protected, but use by any person other than the purchaser or on more than one CPU at any specific time is strictly prohibited.

Warranty

Universal Cross-Assemblers will replace the Cross-8 source disk and/or reference manual once for a period of ninety days from the date of purchase if the original disk and/or reference manual are unreadable upon their return.

Shipping

North American orders will be shipped via First Class Mail, with all others being shipped Air Mail. Universal Cross-Assemblers does not place shipping or foreign order surcharges on delivery of any of its products.

So drop the enclosed order card in the mail and write assembly language programs for almost any 8 bit processor using this revolutionary microprocessor development product. It may be the last microprocessor development software package you will ever have to buy.